December 2006

Parallel Architectures for Searching the MEDLINE Database

Rajendra Boppana
University of Texas at San Antonio

Suresh Chalasani
University of Wisconsin-Parkside

Bob Badgett
University of Texas Health Science Center

Jacqueline Pugh
University of Texas Health Science Center

Parallel Architectures for Searching the MEDLINE Database

Rajendra V. Boppana
Computer Science Department
University of Texas at San Antonio
San Antonio, TX 78249
boppana@cs.utsa.edu

Suresh Chalasani
School of Business and Technology
University of Wisconsin-Parkside
Kenosha, WI 53141-2000
Suresh.Chalasani@uwp.edu

Bob Badgett, M.D.
Division of General Medicine
University of Texas Health Science Center
7703 Floyd Curl Drive
San Antonio, TX 78229
badgett@uthscsa.edu

Jacqueline A. Pugh, M.D.
Professor of Medicine
Division of General Medicine
University of Texas Health Science Center
7703 Floyd Curl Drive
San Antonio, TX 78229
pugh@uthscsa.edu

ABSTRACT

The MEDLINE database, an online archival of medical research publications, is extensively used by physicians, clinicians and other medical researchers for literature search and information on prior case studies. The size of MEDLINE database is over 40GB. The PubMed is an implementation of MEDLINE and provides basic search capabilities freely via the Internet. The initial search often tends to return excessive irrelevant pointers; so researchers often need to conduct additional searches to refine the search data. In this paper, we will investigate a parallel architecture for searching the MEDLINE database, which is unique compared to the existing MEDLINE implementations. The proposed architecture will implement MEDLINE on a cluster of Personal Computers (PCs). This architecture (i) allows for refinement of searches, (ii) reduces the search time for the users, and (iii) increases the availability of the system. We present design guidelines to select the cluster configuration and an analytical model to predict performance improvements.

Keywords: MEDLINE, Parallel Processing, PubMed, Search Algorithms, Cluster of workstations.

INTRODUCTION

Health care providers are awash in publications, textbooks, and guidelines recommending how to assess, diagnose, and treat medical conditions. Yet, even with the advent of the Internet, it is often extremely difficult in clinical practice to find answers in a timely manner to specific questions that arise in the management of patients' health issues. The central repository of medical research is the MEDLINE database. The time needed to search and *review* results from MEDLINE may range from 20 minutes to 2 ½ hours (Lucas et al. 2004); this greatly exceeds the two minutes that clinicians typically have available to answer questions during clinical care (Ely et al. 1999). Consequently, clinicians rarely use MEDLINE (Ely 1999); when they use MEDLINE, they do not use it well (Hersh 1998).

In 1997 the National Library of Medicine launched PubMed which gave the first public and free access to MEDLINE with links to full texts of articles (Anonymous 1997). To reduce the time needed to search and review results, SUMSearch was developed by Badgett et al. (Badgett 2001) to automatically examine the results from an initial MEDLINE search and revise queries multiple times to extract more relevant information. Thus, each query from a user may result in up to 10 queries by SUMSearch of MEDLINE and other resources. SUMSearch is internationally recognized (Glanville et al. 2003, Dearness & Tomlin 2001, Anagnostelis 2002). However, owing to the restrictions on the access to MEDLINE by National Library of

Medicine, the refined queries need to be issued sequentially and rate of queries needs to be throttled. Due to the Internet delays accumulating during multiple searches, SUMSearch's response time to a query can be as much as 30 seconds.

In this paper, we describe a parallel architecture for MEDLINE database integrated with search refinement tools to facilitate accurate and fast response to search requests by users. The proposed architecture, to be developed by the authors, will use low-cost, high-performance computing clusters consisting of Linux based personal computers (PCs) (i) to provide subsecond response times for individual searches and (ii) to support several concurrent queries from search refinement programs such as SUMSearch.

Fast search engines that run on a cluster of PCs and sift through a large volume of content to find relevant information have been implemented by several commercial companies. For example, there are more than eight billion web pages registered with the Google search index as of 2005. Google uses sophisticated and proprietary techniques to find and return relevant internet documents requested by the user. Fast searching can be accomplished by distributing the problem of searching among multiple computers that search simultaneously in different parts of data. Distributed processing and distributed databases have been significant research topics in the years past. However, special techniques to optimize the response time of searching are required. For example, loading data to be searched into memory before any searching begins has been explored (Chalasani and Boppana 2005). Constructing index trees based on keywords to speed up searches has also been explored by several researchers (Melnik et al. 2001, Yu et al. 2003). However, no comprehensive implementation that combines all these different techniques for fast searches has been reported in literature. Most such implementations are commercial and hence proprietary in nature. In this paper, we will combine techniques such as (a) indexing large databases using keywords and content, (b) in-memory loading and searching of databases, (c) distributing content to be searched on multiple computers, and (d) increasing the precision of searches using finite refinements, to achieve fast searching of MEDLINE data.

Although significant advances have been achieved in distributed processing, no guidelines are available for scalable implementations of search engines for large databases. MEDLINE database consists of data in excess of 40GB. Speeding up searches in this database requires careful construction of search trees in memory, searching on and combining results from multiple computers, and pruning of results to improve the precision of searches. Using keywords in the MEDLINE database, index trees will need to be constructed so that keywords and documents associated with these keywords can be randomly accessed instead of sequential access. In addition, the underlying documents must be distributed across parallel computers so that only a portion of the documents resides in the memory of a single computer. In response to the incoming search request with keywords, each computer searches for documents associated with the keywords in its memory. The results of these multiple searches among parallel computers will be combined and returned to the user. This solution, though logical and appears achievable in theory, is difficult to implement in practice because of performance constraints such as achieving sub-second response times. Our research will address the implementation issues for MEDLINE searches in the following specific areas: (a) distribution of documents among parallel computers so that each computer has the same amount of load in response to a "typical" query, (b) retrieving documents from the database quickly once the index trees are searched, (c) refining searches within each parallel computer to improve the precision and accuracy of searches, and (d) optimizing the broadcast and multicast operations that enable parallel computers to combine the search results.

The rest of this paper is organized as follows. Section 2 presents the parallelization of MEDLINE database and an architecture for parallel MEDLINE implementation. It also discusses management of MEDLINE data using multiple index trees. Section 3 presents a mathematical model for reducing the overall search time for MEDLINE data and indicates a formula for the optimum number of database servers. Section 4 provides a model for AutoMedline. AutoMedline is a proposed web-service using which clinicians and researchers can access our parallel MEDLINE implementations via the Internet. Section 5 concludes this paper with directions for further research.

PARALLEL ARCHITECTURES FOR THE MEDLINE DATABASE

PROPOSED MEDLINE DATABASE ARCHITECTURE AND IMPLEMENTATION

The proposed architecture for a parallel MEDLINE implementation is illustrated in Figure 1. This architecture exploits two types of parallelism:

Temporal parallelism: The incoming requests are distributed equally among the application servers (Appservers). An incoming request is completely handled by one single Appserver.

Spatial parallelism: The MEDLINE database is equally distributed among all database servers (DBservers). In response to an incoming search request, the documents corresponding to that search are retrieved from one or more DBservers.

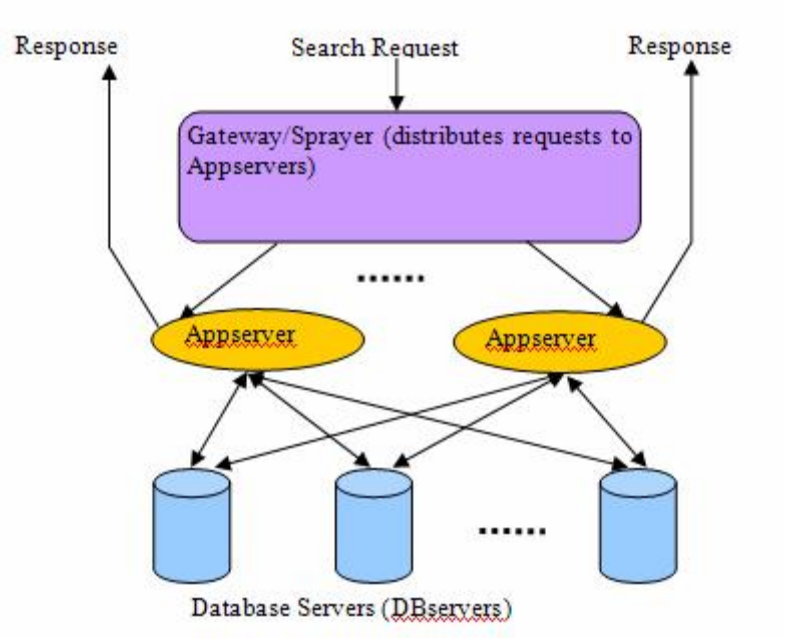

Figure 1. Parallel Architecture of the MEDLINE module. The actual hardware can be a simple fast Ethernet based Linux cluster.

In this architecture, the application server (Appserver) receives the search request. Each application server has the complete index tree for the MEDLINE database. The incoming requests are evenly distributed across application servers by the Gateway server (Sprayer machine). The application server quickly searches its index tree for the keywords contained in the request. The index tree search identifies the documents relevant for those keywords and the database servers (DBservers) on which those documents reside. The Appserver then requests the DBservers for the identified documents. DBservers return the documents to the Appserver, which then combines the results from different keywords and sends a response to the requestor. This procedure is indicated in Figure 2.

We need to address three critical issues for successful implementation of MEDLINE and to facilitate subsecond search times: (i) implementation of the search engine, (ii) implementation of the database, and (iii) automatic revision of searches. These are further discussed in the next three subsections.

To facilitate fast and multiple concurrent searches, index lists are often created for a database. An index list is similar to the index at the back of a textbook and indicates the IDs of records that contain a given word. For example, when a request for a list of documents containing the words "hypertension" and "diabetes", is submitted, the index lists for these two words will be examined to identify records that contain both words. (Alternatively, a document vector that indicates words (in dictionary order) contained in a document can be used. Given the large vocabulary used in medical publications, the index list approach is preferable for this implementation.) The records so identified will be retrieved from the DBservers and composed to generate a response. Figure 3 shows an example index list in the form of a tree search structure. Each keyword (or search term) is a node in this tree. Each node is associated with a list of the database servers and the document ids on that database server that point to relevant documents for that keyword. In the example shown in Figure 3, the relevant documents for the term “diabetes” are documents with IDs 2, 100 and 215 on DBserver 1 and documents with IDs 625, 901 and 1576 on DB server 5. Even though this example indicates a tree search mechanism (to simplify the discussion), the implementation in reality will follow a hash-map implementation so that searching for a keyword takes almost a constant amount of time, regardless of the search term.

Index lists will be created for the most frequently occurring words and phrases in the database. These index lists will be created at one time and stored (replicated) in the main memories of Appserver machines, which process the searches. We will write a multithreaded program to examine the index lists and identify the records that satisfy the search request. With multithreading we can take advantage of multiple CPUs in each machine. To handle multiple concurrent searches we can simply increase the number of Appservers.

Process Steps for Servicing a MEDLINE Search Request by the Parallel Architecture

// Input to this process is the MEDLINE Search request including all the keywords.

Step 1: Gateway server (Sprayer machine) identifies an Appserver and forwards the search request to the Appserver.

Step 2: Appserver searches its copy of the index tree for each keyword and identifies the DBservers on which the corresponding documents reside. Appserver sends requests for those documents (it sends a unique request id, the keyword and the unique document ids for that keyword to the identified DBservers).

Step 3: Each DBserver which receives a request for the documents with document ids reads the corresponding documents from its memory and sends those requests to Appserver.

Step 4: The Appserver combines the resulting documents supplied by the DBservers and sorts the results by relevance. The Appserver then decides whether the search needs to be further refined; if so, it sends another set of retrieval requests to the DBservers. If the search does not need to be refined, it sends the results to the requestor.

Figure 2: Process steps for handling a single MEDLINE search request.

SEARCH ENGINE IMPLEMENTATION

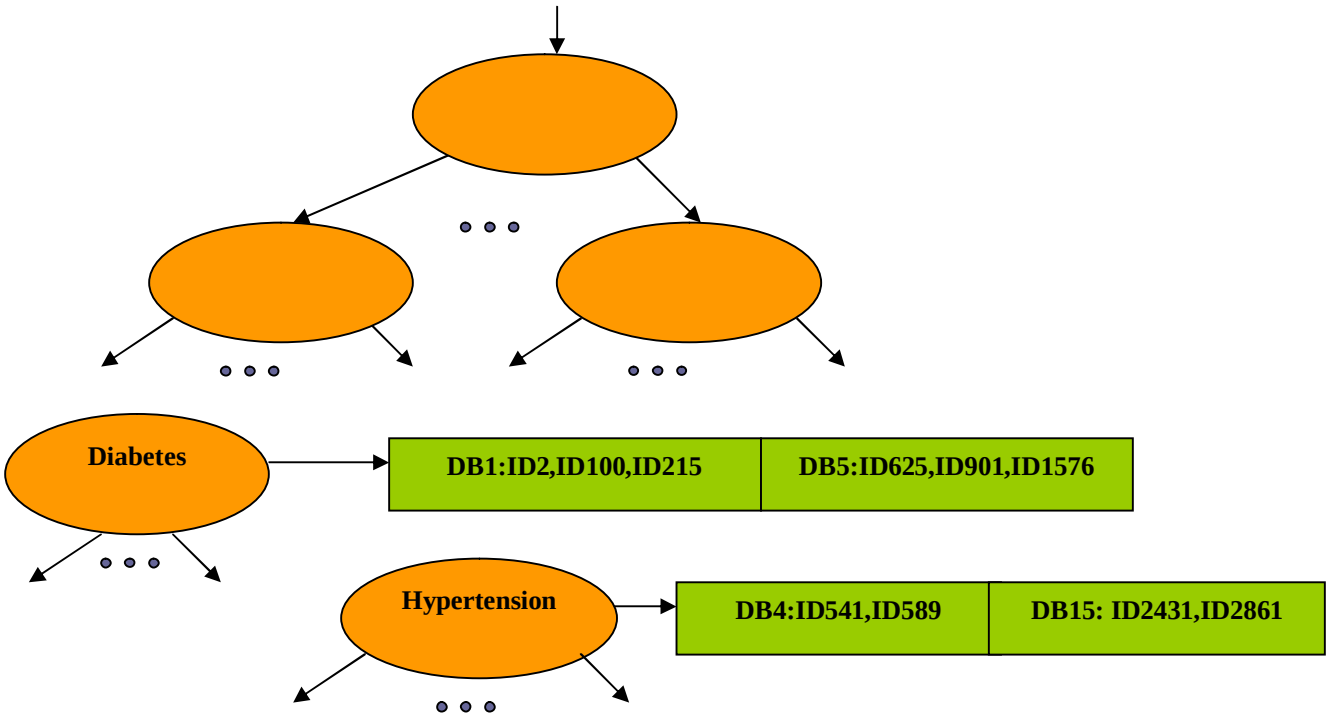

Figure 3. An example index list in the form of a search tree.

From the user's standpoint, a query is submitted to a web server. For load balancing purposes, one machine can act as a sprayer or distributor of requests (see Figure 1) and distribute the request to an available Appserver. The web server, which contains the controller module, acts as a conduit to pass the query request to a common gateway interface (cgi) program, programmed in C. This program will send the search request to an Appserver ensuring that all Appservers are evenly loaded with search requests. An Appserver in turn searches the index lists, retrieves appropriate records from DBservers. Then the Appserver examines the search results and combines it with results from other sources to format a response to the user. If necessary, to refine the search results, the Appserver will generate new search requests and processes these new requests before it sends a final response to the user. This process is illustrated in Figure 2.

The size of the index list can easily fit in a few GBytes of memory. With 40 GB of data, it is reasonable to assume that there are a million documents in the MEDLINE database. Let us assume that there are 100,000 keywords used to search the database. Of these 10,000, may occur frequently in the database. Such keywords are indexed by a binary string to indicate their presence in each document by a 1 or 0. Each such keyword requires a million bits or 128 KB of memory. In addition, the other less frequent keywords may occur in an average of 1000 documents. In that case, each of the less frequent keywords requires 8,000 bytes (assuming each database record's unique located id is 8 bytes long). The amount of memory required for all 100,000 keys is less than 800 MB. For the 10,000 most frequently keywords, we will need 10000 * 128 KB = 1.28 GB of memory. So the total memory requirements for the keeping the index list in the main memory is slightly over 2 GB. This is well within the scope of a moderately priced workstation.

DATABASE IMPLEMENTATION

The database will be implemented using either the MySQL or PostGreSQL database engines. Both engines are high-performance engines which support multi-processor hardware platforms. The databases can be searched using SQL syntax queries. To facilitate subsecond search times, we need to avoid searching the database from disk memory directly. To achieve this, we will split the database into nearly equal parts and allocate each part to a distinct machine (denoted, DBserver). We will develop software that will let each DBserver (at boot time) to read the database and store the records as Java objects in main memory. We will create a hash (mapping) mechanism to associate these Java objects with the unique IDs of the records they contain. To reduce the memory requirements, we will consider compressing infrequently used records. To improve performance, most of the data from the database will be loaded into the DBserver's memory. If there are 10 DBservers, each DBserver needs to load 4GB of data into its memory (to load the complete 40GB MEDLINE database). DBservers with 8GB of memory can be purchased at a cost of $10,000 per server.

The MEDLINE database can be divided among the DBservers in two different ways as discussed below.

Strategy 1: Distribute MEDLINE data equally among all DBservers.

Strategy 2: Distribute MEDLINE data among DBservers so that the overall load is approximately equally distributed at any given time.

Strategy 2 has the potential to improve the performance of our parallel implementation by distributing the load evenly among all DBservers. To implement this strategy, however, we need to estimate the probability with which each document in the MEDLINE database is accessed. For example, if document D_i is accessed with a probability of p_i, MEDLINE database documents need to be distributed among n DBservers such that the combined probabilities of all documents on any given DBserver is approximately 1/n. Probability p_i with which document D_i is accessed can be computed by looking at access trends of documents over an extended period time. This strategy is difficult to implement in practice compared to Strategy 1.

AUTOMATED REVISION OF SEARCHES

We will develop a controller module that will direct revisions to searches. The controller will submit the user's query to the database and examine the number of citations retrieved by each search. If more that 50 articles are returned (Blair 1980), the controller will progressively add more restrictive search limits. The progressive limits will be based on the ones used by SUMSearch and also the search filters developed by the Hedges Team (http://hiru.mcmaster.ca/hedges/). For example, initially search terms will be searched in the title, abstract and MeSH fields. If excessive citations are returned, the controller can revise and require search terms to be either title words or MeSH major terms. As a second example, if users choose to focus their search with one of the filters of the Hedges Team, the initial filters will be based on the 'sensitive' filters from the Hedges Team. If this retrieves excessive citations, the search will be resubmitted with the 'specific' filter.

The final search results are formatted in XML with the following information.

(1) *Summary header data.* Includes the name of any clinical filter applied, if requested by the user (http://hiru.mcmaster.ca/hedges/), and the publication date of most recent articles retrieved.

(2) Number of citations found with each search.

(3) The URL to the abstract at PubMed for each article retrieved in the final search.

We will evaluate the feasibility of including in the XML a short summary of each article. The summary would be parsed from the MEDLINE data and would be the shorter of either the last section of a structured abstract or the last two sentences of the abstract.

PERFORMANCE CONSIDERATIONS

As the number of users increase, more web servers and Appservers can be added to handle multiple concurrent search requests. On the other hand, if the request load is not expected to be high, then we can combine the functionality of a web server and Appserver into a single machine. We believe that our approach provides flexibility to scale up to handle larger databases and larger request volume if needed and to scale down to facilitate a low-cost implementation if search time constraints are not stringent. We will use Linux operating system with public domain MPI and OpenMP software packages to use multiple machines and multiple CPUs in each machine efficiently. For database implementation, we will use MySQL or PostGreSQL to implement DBservers. We will use Java and C to develop the search engine and other custom software as needed.

We will monitor the resulting MEDLINE webservice by observing its response times in response to queries from the search engine to the MEDLINE gateway. In addition to recording search times of the naturally occurring searches submitted by users, we will also submit automated requests every hour. The automated requests will be important during development when there are few requests for searches.

In addition, the static data from the MEDLINE database will be loaded into memory when the application starts and before any user request is processed. Our software will load the static data (data that does not change frequently) at the beginning of the application and stores them in memory cache. As discussed in Section 2, we expect most of the data including the MEDLINE data and the index trees to reside completely in the memory; hence, no penalty for disk-reads is incurred in the parallel MEDLINE architecture.

MODEL FOR PREDICTING SPEEDUP

In this section, we present a simple analytical model to determine the optimal use of the cluster to speedup a single MEDLINE search. The response time of a query can be broken up into several components: (a) the time taken for a search request to reach an Appserver from a user machine, (b) the time taken for an AppServer to broadcast the search information to the DBservers, (c) the time taken by DBservers to send data back to the AppServer, and (d) the time taken by the Appserver to format and send the results back to the client machine. Of these, the time for tasks (a) and (b) are very small and are nearly constant for all searches; we can ignore these times in our model.

Let M be the average amount of data in bytes resulting from a single search. Let $M*t_a$, where t_a is the data format time per byte of data, be the amount of time it takes for an AppServer to format and send data to the client machine. Let $M*t_r$, where t_r is the retrieval time per byte of data, be the amount of time it takes to complete the search using a single processor. If n DBservers are used to perform the search, then the search time is reduced to $(M*t_r)/n$. But this data need to be transferred from the DBservers to the AppServer. On the average, each DBserver transmits M/n bytes of data.

The communication among servers in a cluster can be modeled as $t_s + m*t_b$, where t_s is the start-up time required to format and prepare transmission of a message, t_b is the transmission time per byte of message, and m is the number of bytes transmitted [Grama et al. 2003]. So the total time taken to process a query, T_{search}, is the sum of the search time, the data transfer time, and the data format time.

$T_{search} = (M*t_r)/n + n*[t_s + (M/n)*t_b] + M*t_a = (M*t_r)/n + n*t_s + M * t_b + M*t_a$.

To minimize T_{search}, we need to differentiate the right hand side of the above equation with respect to n, equate it to 0, and solve it for n.

$-(M*t_r)/n^2 + t_s = 0$

So, $n = (M*t_r/t_s)^{1/2}$ gives the optimal number of DBservers to be used to satisfy a single query.

The message startup time and computational time per unit data are dependent on the technology used. With fast Ethernet and standard Linux network protocol stack with parallel processing middleware such as MPI (message passing interface) [Gropp et al. 1999], the startup time is about 100 microseconds. The retrieval time can be 100 nanoseconds (sufficient to execute about a thousand CPU instructions) or more per byte of data. If the average amount of data produced in a search is 100 Kbytes, then the optimal number of DBservers to be used is at least

$(10^5 * 10^{-7}/10^{-4})^{1/2} = (100)^{1/2} = 10.$

If the message startup overhead is reduced, for example, using more expensive Infiniband or Myrinet interconnects with optimized network software [Liu et al. 2003], then more DBservers can be used efficiently. On the other hand, if several queries need to be handled simultaneously, then more queries can be completed per unit of time if the number of DBservers is allocated proportionately to the amount of data to be generated by each query. We will incorporate this model to dynamically vary the number of DBservers used for efficient and fast completion of search requests.

AUTOMEDLINE: WEBSERVICE IMPLEMENTATION OF PARALLEL MEDLINE

We anticipate that a responsive MEDLINE search engine will be used by clinicians and researchers extensively. Therefore, we intend to provide access to the MEDLINE database via the campus intranet initially and via the Web eventually to everyone else.

The Internet presence includes a website that displays the user interface and also a web service that exposes the MEDLINE database to remote searching by authorized collaborating websites. To facilitate easier user interface, we will incorporate user selectable clinical filters developed by other researchers in the medical field. The user may also access the MeSH browser to look up the canonical search term. Lastly, the user will have the option of turning off automation if the user wants more control over the search strategy and making revisions.

The web site will direct the user's query to the database and will receive the results from the database in XML. Scripting at the website will transform the XML into html that is returned as a web page to the user. The web page of results that the user receives contains links to the full texts of the articles at the publishers' websites.

To facilitate automated, high volume queries by licensed third parties, we will also provide a direct query mechanism without going through the web interface. In response to a query to the webservice, the search results will be sent as XML documents to allow the remote website to insert the results into its webpages directly.

DISCUSSION

The National Center for Biotechnology Information (NCBI) currently provides access to the MEDLINE database using the PubMed interface. Access to PubMed and other NCBI databases are served by multiple webservers to achieve load balancing. Since requests to all NCBI services go through the same web servers, the NCBI architecture is not designed to exploit the inherent parallelism of the MEDLINE database. The emphasis of the NCBI architecture is to provide good response times for a variety of NCBI services including MEDLINE and GenBank. Our approach focuses entirely on the MEDLINE database and exploits spatial and temporal parallelism that is inherent in the MEDLINE database.

To implement the parallel MEDLINE architecture, the required hardware cost is estimated as follows. A dual or quad multiprocessor machine with multiple network cards at a cost of $8,000 may be used as the gateway server/sprayer. Two to four multiprocessor machines at a cost of $6,000 can used as application servers. The 10 database servers with 8 GB memory will cost $10,000 a piece. The interconnection of these machines can cost $100 (gigabit Ethernet) to $1,000 (Infiniband) per machine. The total hardware cost is at about $140,000 even with special high-speed interconnect. However, this hardware cost is relatively minimal if the parallel MEDLINE, because of its fast search capabilities, can attract clinicians to use it for patient care.

The creation of a webservice for AutoMedline allows integration of parallel MEDLINE searches into other web-sites. Although the standalone website for AutoMedline will greatly help searchers of MEDLINE, the webservice allows additional opportunities. Although many online medical resources have appeared, we believe no single resource is adequate to answer all clinical questions. The creation of a webservice allows integration into the online medical resources and search engines of third parties.

Further research on this topic can be conducted in several different directions. First, results from a prototype implementation of the parallel MEDLINE architecture can be reported. Second, cost-benefit and ROI analysis of such implementations can be studied in detail to examine whether hospitals and clinics will be interested in local copies of such parallel MEDLINE systems or subscriptions to webservices such as AutoMedline. Another direction is to enhance the search speed using the

most commonly used search terms and phrases; this can be accomplished by constructing the complete results for most commonly used terms and storing them in Appserver memory so that they can be readily retrieved in response to the search request.

Another interesting direction is to integrate the parallel MEDLINE design with other databases dispersed geographically. Given the response time constraints and relatively small size of the database, grid based implementation MEDLINE is not attractive. However, distributed computing techniques developed for grid computing (Parashar and Lee 2005) will be useful in integrating our MEDLINE implementation with other database search engines over the Internet.

CONCLUDING REMARKS

This paper presented a parallel architecture for speeding up MEDLINE searches. The parallel MEDLINE architecture is based on constructing index trees for the MEDLINE database and storing this index tree in the memory of application servers. The index tree is replicated among all application servers and the incoming requests for MEDLINE searches are evenly distributed among all application servers. However the MEDLINE database is distributed among all database servers (or DBservers). In response to a search request, the application server (or Appserver) searches the index tree for keywords, identifies the DBservers on which the corresponding documents reside. The Appserver sends a request to read these documents to the DBservers and gathers the resulting documents from the DBservers and compiles a response. In some cases, the Appserver may further refine the searches based on the documents retrieved. This paper presented an analytical model on the number of database servers needed and indicated techniques for performance improvement.

ACKNOWLEDGEMENTS

Boppana's research has been supported by the NSF (National Science Foundation) grants EIA-0117255 and ANI-0228927 and the San Antonio Life Sciences Institute grant 10001642. This research is also sponsored in part by a summer research grant to Suresh Chalasani from the University of Wisconsin system. Bob Badgett and Jacqueline Pugh's research is supported in part by the VERDICT Research Enhancement Award Program, South Texas Veterans Health Care System Department of Medicine and the University of Texas Health Sciences Center. The views expressed in this article are those of the authors and do not necessarily represent the views of the Department of Veterans Affairs.

REFERENCES

1. Lucas BP, Evans AT, Reilly BM, Khodakov YV, Perumal K, Rohr LG, Akamah JA, Alausa TM, Smith CA, Smith JP. (2004) The Impact of Evidence on Physicians' Inpatient Treatment Decisions. *Journal of General Internal Medicine*, May 2004, 19(5p1):402-9.

2. Ely JW, Osheroff JA, Ebell MH, Bergus GR, Levy BT, Chambliss ML, et al. (1999) Analysis of questions asked by family doctors regarding patient care. *British Medical Journal*, Aug 1999, 319 (7206):358-61.

3. Hersh WR, Hickam DH. (1998) How well do physicians use electronic information retrieval systems? A framework for investigation and systematic review. *Journal of the American Medical Association (JAMA)* Oct 1998, 280(15):1347-52.

4. Anonymous (1997) Free Web-Based Access to NLM Databases. *NLM Technical Bulletin*, May-June1997,296.

5. Chalasani S and R. Boppana. (2005) Software Architectures for E-Commerce Computing Systems with External Hosting. *International Journal of Computers and Applications*, vol. 27, no. 3, pp. 190-198, 2005.

6. Melnik S, Raghavan S, Yang B, Garcia-Molina H. (2001) Building a Distributed Full-Text Index for the Web [Web Page].; Available at http://www-db.stanford.edu/~melnik/pub/melnik_TOIS01.pdf. *ACM Transactions on Information Systems* (TOIS). Vol. 19, No. 3, pp. 217-241.

7. Yu C, Cuadrado J, Ceglowski M, Payne JS. (2003) Patterns in Unstructured Data [Web Page]. Available at http://javelina.cet.middlebury.edu/lsa/out/cover_page.htm. (Accessed Jan 15 2005).

8. Blair D. (1980) Searching biases in large interactive document retrieval systems. *Journal of American Society of Information Science and Technology* 1980; 31:271-77.

9. Glanville J, Wilson P, Richardson R. (2003) Accessing the online evidence: a guide to key sources of research information on clinical and cost effectiveness. *Quality & Safety in Health Care*, Jun 2003;12(3):229-31.

10. Dearness KL, Tomlin A. (2001) Development of the National electronic Library for Mental Health: providing evidence-based information for all. *Health Information & Libraries Journal*, Sep 2001;18(3):167-74.

11. Anagnostelis B. (2002) A guide to healthcare resources on the Internet. *Health Information & Libraries Journal* March 2002;19(1):59-60.

12. Badgett RG, Paukert JL, Levy L. (2001) The evolution of SUMsearch for teaching clinical informatics to third-year medical students. *Academic Medicine*, May 2001;76(5):541.

13. Gropp W, Lusk E, Skjellum A. (1999) Using MPI: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). 2nd Edition. Boston, Massachusetts: MIT Press, 1999.

14. Grama A, Gupta A, Karypis G, and Kumar V. (2003) *Introduction to Parallel Computing*, 2nd Edition, Addison Wesley, 2003.

15. Liu J, et al. (2003) Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics, p. 58, *ACM/IEEE Supercomputing*, 2003.

16. Parashar, M and Lee, C (2005) *Proceedings of IEEE, Special Issue on Grid Computing*, March 2005, 93(3):479-484.